第3章

X-by-wireシステムのためのソフトウェア設計

――タイム・トリガOSやミドルウェアを利用したFlexRayネットワークの構築

服部博行

自動車の軽量化，それに伴う燃費向上，デザインの自由度の拡大といった利点からX-by-wire技術が注目されている．X-by-wire技術の適用箇所としてはステアリング（Steer-by-wire），スロットル（Throttle-by-wire），ブレーキ（Brake-by-wire）などが考えられている．筆者ら（ヴィッツ）は，これら三つの“by-wire”アプリケーションを実現するデモンストレーション・システムを製作した．本システムには，TOPPERSプロジェクトから公開されているFlexRay通信用のオープン・ソフトウェアが組み込まれている．ここでは，このデモンストレーション・システムをもとに，ノード間の通信やスケジュール・テーブルの記述方法などを解説する．なお，ここで紹介するシステムのデモンストレーションのようすは，ビデオ・データとして本誌付属CD-ROMに収録されている．（編集部）

現在，標準的な車載ネットワークとしてCAN（controller area network）が利用されています．今後，用途によってはデータ転送速度や安全性の面でCANの能力不足を補う必要性が出てくると考えられています．そうした用途に特化した車載ネットワーク規格として，欧州の自動車メーカや電装メーカを中心にFlexRayが策定されています．

FlexRay通信は，パワートレイン（動力伝達）系やシャーシ（車両）系などでの採用が検討されている“X-by-wire”への適用を想定しています．X-by-wireは，機械的な装置の間のつながりを通信信号（電気的配線）に置き換える技術です（**図1**）．X-by-wireを用いた通信システムは，今まで以上の信頼性や応答性を必要とします．

● 電気信号でハンドルやブレーキを操作するということ

X-by-wireは，油圧やシャフトなどの機械的な機構を用いずに，電気的にステアリング操作やブレーキ操作などを

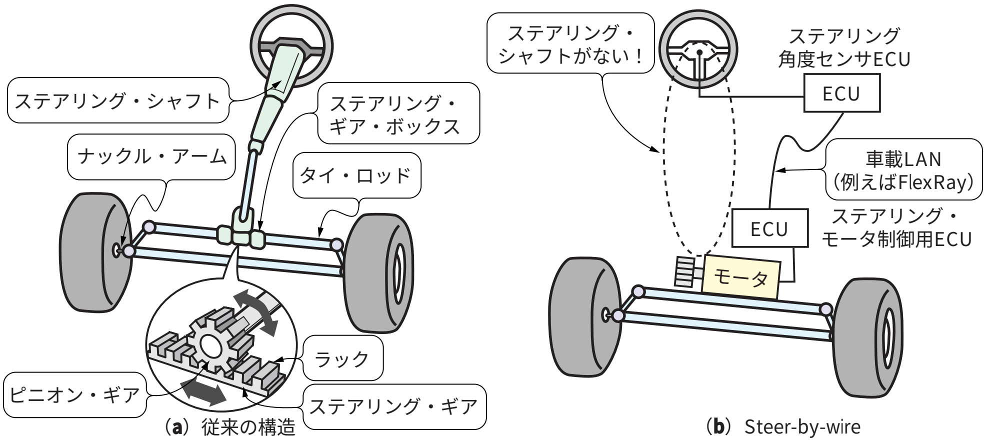


図1 Steer-by-wireの例[9]
（b）のSteer-by-wireでは，ユーザ・インターフェースであるハンドルとステアリング部が機械的につながっていない．つながっているのはネットワーク通信路，いわゆる“wire”のみ

KeyWord X-by-wire，TOPPERS，CAN，FlexRay，タイム・トリガ，TT-OS，OSEK/VDX，TT-COM，FlexRay-NM，FlexRay-DRV

行う技術です．機械部品が省略できるため，設計自由度が上がります．また，部品の組み付けコストの削減や車体の軽量化なども見込めます．これによって，車両の製造コストを削減したり，燃費を向上させることなどが期待できます．さらに，ソフトウェアだけで高度な車両制御を行えるようになるので，自動運転などを実現するための基本技術ともいえます．

現在の車の多くにThrottle-by-wireが利用されています．いわゆる「電子制御式スロットル」と呼ばれるものです．運転者のアクセルの踏みぐあいをセンサで読み取り，その情報がネットワーク（もしくは直接的な接続）を介してECU（電子制御ユニット）に伝達され，エンジンのスロットル・バルブの開閉を制御します．電子制御式スロットルが利用される以前は，アクセル・ペダルとスロットル・バルブはワイヤ[注1]で接続されていました．アクセルの動作に連動してスロットル・バルブが引っ張られ，バルブの開閉を行っていました．この方法では，ECUが介在する部位がなく，現在の車に搭載されている姿勢制御（路面の状態に応じた自動車の安定走行の制御）などは実現できません．

注1：ここでいう「ワイヤ」は，機械部品のワイヤを意味する．一方，X-by-wireのwire は通信経路としてのワイヤ・ハーネスなどを示している．
注2：ここでいう「時間」は，ECUが個別に管理している時間ではなく，もっと広い世界（例えばネットワーク上の時間など）で管理された時間を意味する．

また，展示会のデモンストレーションなどでは，Steer-by-wireが多く取り上げられています．これは視覚的にX-by-wireのメリットが見えやすいからです．例えば，機械的な制限がないため，右ハンドルの場合と左ハンドルの場合の設計が容易であったり，運転席を自由に配置できるというメリットがあります（写真1）．

1 FlexRay通信に必要なソフトウェアとは

汎用OSやリアルタイムOSの多くは，「イベント・ドリブン（事象駆動）型」の制御に対応しています．イベント・ドリブンとは，なんらかのイベントが発生したときにそれに応じた処理を行う方式です．例えば，「ボタン1を押したら（イベント発生），赤いランプが点灯する（処理実行）」といった制御をイメージしてください．

一方，タイム・トリガ（時間駆動）型OSでは，外部からのイベントなどではなく，時間的[注2]な要因が処理要求となり，それに対応する処理を実行します．例えば，「5分ごとに（時間的要因），赤いランプの点灯/消灯（処理実行）を繰



写真1　実用化が見えてきたX-by-wire
自動車関連の展示会では，最近，X-by-wireを採用したコンセプト・カーが展示されるようになった．写真は，「第39回 東京モーターショー」で展示された日産自動車のコンセプト・カー「pivo」．X-by-wireを用いることで，運転席を回転できる．写真は本誌編集部が撮影

column　ツール・チェーンとOIL

本文でも説明しましたが，FlexRay通信用ソフトウェア（TT-OS，TT-COM，FlexRay-NM，FlexRay-DRV）は，当該ノード内のソフトウェア群になります．そのため，FlexRay通信用ソフトウェアで提供するシステム・ジェネレータ[注A-1]はノード単位からサポートし，FlexRayネットワーク全体をサポートするわけではありません．

ネットワーク全体の通信スケジュールなどには，別途，上位のツールが必要だと思います．こうしたツールは，欧州のツール・ベンダが盛んに開発をアピールしています．多くのツールは「Fibex」と呼ばれる中間言語をサポートするようです．FlexRay通信ミドルウェアもノードごとに分割された情報（Fibex記述）から，OILもしくは直接コードを生成する必要があるのかもしれません．

注A-1：システム・ジェネレータは，システムの整合性チェックやシステム情報の整理といった，本来OS側で行うべき処理を担当する．これによりOSの処理は大幅に省略され，性能が向上する．

り返す」といった制御です．

筆者ら（ヴィッツ）は，FlexRay通信を実現するためのソフトウェアを開発しました注3（開発の経緯については参考文献(5)を参照）．このソフトウェアは，FlexRay通信に必要なタイム・トリガ処理注4を実現するOS「TT-OS（Time Triggered Operating System）」とFlexRay通信ミドルウェアからなります（**図2**）．また，FlexRay通信ミドルウェアは，「TT-COM（Time Triggered Communication）」，「FlexRay-NM（Network Management）」，「FlexRay-DRV（Device Driver）」から構成されます．

● FlexRayの「通信」と「制御」の両立を目ざしたOS

TT-OSは，OSEK/VDX仕様OS（以下，OSEK OSと略す）とTTM（Time Triggered Module）で構成されます．OSEK/VDXはECU間のインターフェースに関する仕様であり，OSだけでなく通信（OSEK COM）やネットワーク管理（OSEK NM）についても定義しています．またOSEK/VDXは，仕様を定めている欧州の自動車メーカを中心とした仕様策定団体の名称でもあります．

TT-OSは，FlexRay通信の制御方式であるタイム・トリガ（時間駆動）型に対応したOSです．ただし，本OSでは，イベント・ドリブン型の処理も行えるように仕様を策定しています．このことは，「通信」か「制御」かで処理の優先順位をつけるのではなく，自動車というシステム全体で処理の優先順位をつけていることを意味します．

CANベースのタイム・トリガ通信注5を実現するために，OSEK/VDXはTime-Triggered Operating System（以下，OSEKtimeと略す）というOS仕様を策定しています．OSEKtimeでは，イベント・ドリブン処理を担うOSEK OSがタイム・トリガ処理を担うOS（つまりOSEKtime）の上に配置されています（**図3**）．このことは，OSEKtimeが絶対的な権限を持っていることを意味します．すなわち，OSEKtimeではタイム・トリガ処理が優先され，通常のイベント・ドリブン処理はタイム・トリガ処理が実行されていない間だけ行われることになります．そのため，自動車にとって重要なエンジン制御などの車両制御処理が，FlexRay通信に阻害される危険性があります．

TT-OSは，OSEK OSのみを利用しています．タイム・トリガ処理をTTMが担い，OSEK OSの上にモジュールとして搭載しています．TTMは，OSとしての機能を提供するのではなく，OSにタスクの起動などの処理を要求するだけです．そのため，TT-OSで扱うタイム・トリガ処理タスクとイベント・ドリブン処理タスクは，OSから見て差がありません．あくまでも静的に割り当てられた優先順位に従ってタスクの起動順位が決定されるため，OSEKtimeの「制御より通信処理の優先度がつねに上位である」という問題点を回避できます．

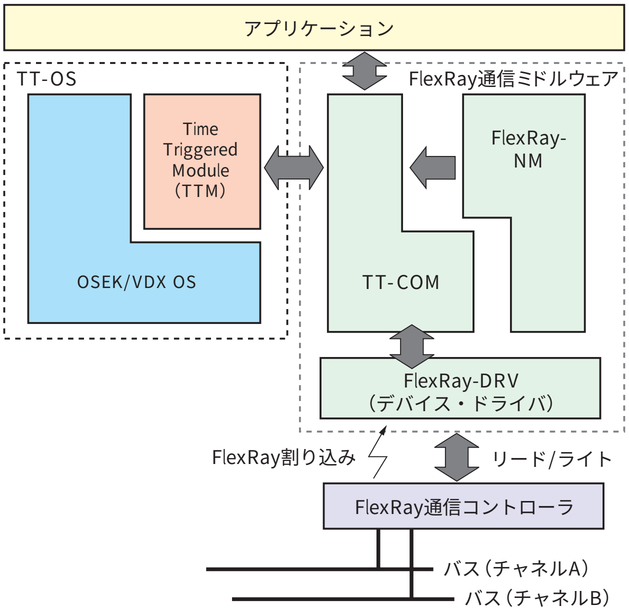


図2　FlexRay通信のためのソフトウェア構成
FlexRay通信ミドルウェアは，OSEK COMと同等の処理を行う通信制御ミドルウェア（TT-COM），ネットワーク管理ミドルウェア（FlexRay-NM），デバイス・ドライバ（FlexRay-DRV）からなる

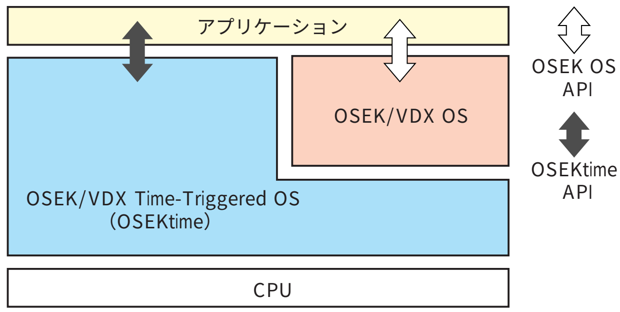
3

● FlexRay通信用のミドルウェアも開発

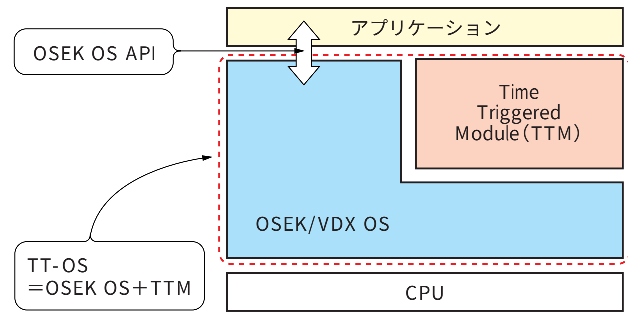
上述したように，FlexRay通信ミドルウェアは三つのソフトウェア部品で構成されています．

- **TT-COM（Time Triggered Communication）** ―― アプリケーション・ソフトウェアからデータの送受信処理を直接指令される通信モジュール（OSEK OSが利用するOSEK COMと同類の処理を行う）
- **FlexRay-NM（Network Management）** ―― ネットワークに接続されているノードの状況やノードのウェイクアップ/スリープなどの管理を実施するFlexRay通信用のネットワーク管理モジュール

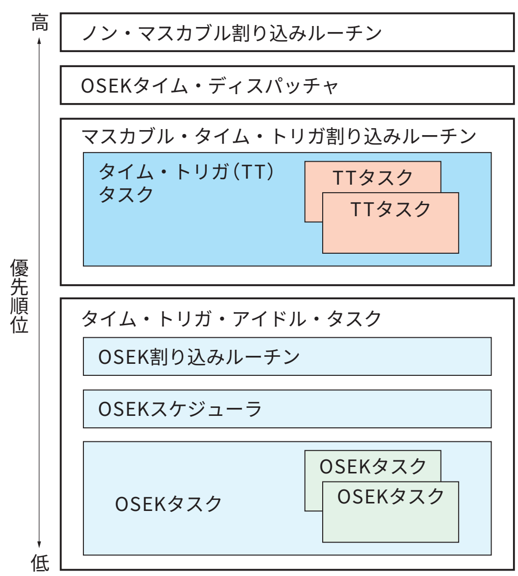
注3：このソフトウェアは，2006年3月からTOPPERSプロジェクト（http://www.toppers.jp/）の会員向けに早期リリースしている．
注4：ちなみに，CANはイベント・ドリブン通信である．
注5：通信デバイスにCANを利用したタイム・トリガ通信として，TT-CAN（Time-triggered Communication on CAN）やTTP/C（Time Triggered Protocol/class C）などがある．



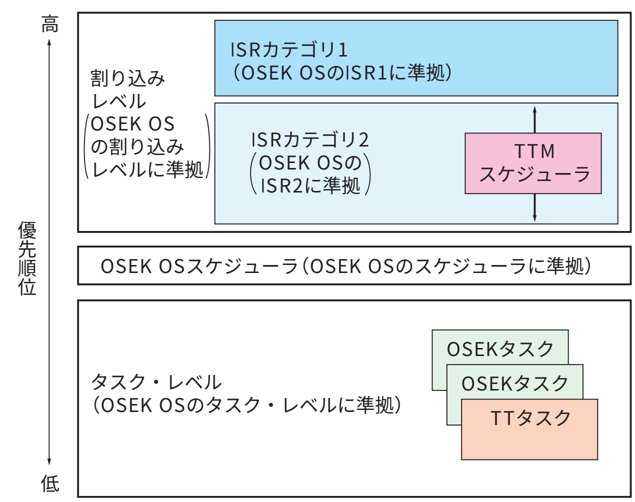
(a) OSEKtimeの構成



(c) TT-OSの構成



(b) OSEKtimeのプロセッシング・レベル



(d) TT-OSのプロセッシング・レベル

図3 OSEKtimeとTT-OSの違い

(a)のように，OSEK OSはOSEKtimeのサブシステムに相当する．そのため，(b)に示すように，OSEKtimeでは非タイム・トリガ(イベント・ドリブン)の制御処理は，つねにタイム・トリガで実行される処理よりも後回しにされる．一方，TT-OSでは，イベント・ドリブン型のタスクとタイム・トリガ型のタスクのどちらもOSEK OSの管理下にあり，タスクの起動順序はタスクに与えられている優先度で決まる

● **FlexRay-DRV(Device Driver)** ―― FlexRayデバイスを操作するデバイス・ドライバ

2 X-by-wireシステムを作ってみよう

筆者らは，上記のFlexRay通信ソフトウェアを利用した簡単なX-by-wireアプリケーションのデモンストレーション・システムを構築しました．ここでは，本システムを例に挙げながら，ハードウェア構成およびソフトウェアの構築方法などを説明します．

X-by-wireのアプリケーションとして，Steer-by-wire，Throttle-by-wire，Brake-by-wireなどが想定されています．今回作成したシステムは，これら3種類の"by-wire"を実現しています．

● 三つのノードを使うわけ

今回利用するFlexRay通信ソフトウェアは，ルネサス テクノロジの協力を得て開発しています．そのため，現時点(2006年5月現在)でこのソフトウェアが動作可能なターゲット・ボードは，ルネサス テクノロジ製のFlexRay評価ボード(M3A-0847G21)となります．今回のデモンストレーション・システムに使用した機材や開発環境を**表1**に示します．今回は，3枚の評価ボードを利用します．この3枚のボードは，それぞれFlexRay通信の各ノードに相当します．

FlexRay通信はネットワーク上で時間を管理しています．これをネットワーク全体で共通の時間(グローバル時間)として各ノードが利用し，同期して動作します．

そのため，ネットワークに接続されたノード間で時間の同期を行う場合，最低二つのノードがあればグローバル時

表1　デモンストレーション・システムに使用した機材と開発環境

	名　称	製品名	メーカ
機　材	FlexRay評価ボード	M3A-0847G21	ルネサス テクノロジ
	ラジコン・シャーシ	F201CHASSIS	タミヤ
	入力装置	HKT-7400（ドリームキャスト用レーシング・コントローラ）	セガ
	モータ・ドライバ	CPU基板「HSB8S2636Q」のサブボード（オプション）	北斗電子
	サーボ・モータ	——	フタバ模型
	そのほか	●評価ボード接続用ケーブル（3ノード・タイプ）2本 ●改造用ピンなど	
開　発 環　境	コンパイラ	NC308 Version 5.20 Release 02	ルネサス テクノロジ
	デバッガ	KD3083 Version 3.30 Release 1	ルネサス テクノロジ
	簡易エミュレータ	M3A-0655 FoUSB	ルネサス テクノロジ



間を作ることができます．ところで，FlexRay通信ではミッション・クリティカルな（リアルタイム性が高く，ある機能の障害がシステム全体の障害につながるような）制御への適用を考え，二つの通信チャネルを利用します．接続ハーネスの断線など，故障や障害への耐性を向上させるために冗長化しているのです．同じことがグローバル時間を作成する場合にも当てはまります．ネットワークを構成するノードが2ノードの場合，どちらかのノードに障害が発生すると，ネットワーク全体の障害となり，正常なノードも動作できなくなります．そこで接続ノードを三つにした場合を考えます．このとき，いずれか一つのノードに障害が発生すると，たとえそのノードがグローバル時間を生成するノード（FlexRay仕様ではコールドスタート・ノードという）であったとしても，直ちにネットワークから切り離され，ほかの二つのノードでグローバル時間が作成されます．この場合，障害発生ノードは動作できませんが，正常動作中のノードに悪影響を与えません．このような理由から，本デモンストレーション・システムでも3ノードによるFlexRayネットワークを実現したいと思いました（なお，国内の車載電子制御の標準化組織であるJasParは，最低4ノードを推奨しているもよう）．

● 三つのノードに処理を割り当てる

今回のデモンストレーション・システムでは，入力装置としてドリームキャスト用のコントローラを利用します．このゲーム・コントローラから，操舵角やアクセル，ブレーキに関する情報を読み出します．これらの情報収集を行うため，三つのノードのうちの一つを割り当て，ゲーム・コントローラに接続します．

次に，動作対象について考えます．電動ラジコンのステアリング装置は，ラジコン用のサーボ・モータとステアリング稼働部が接続されており，サーボ・モータを制御することによりステアリング操作が可能となります．一般的なラジコン用サーボ・モータは，パルス幅制御（PWM：pulse width modulation）で制御します．モータを駆動させるにはラジコン用のアンプを利用することも考えられますが，今回は北斗電子製のモータ・ドライバを用いました．このモータ・ドライバは，ラジコン用モータ・コントローラと同じように，PWM制御によってモータの正転/逆転や回転数を制御します．

また，電動ラジコンにはブレーキ機構がありません．このような場合，モータの両極をハイ・インピーダンス出力にすることでモータ・ブレーキを利用できます．今回のシステムではこの特性を利用しました．

こうしたモータの制御を一つのノード（動作制御ノード）に担当させます．

残り一つのノードがとくに処理を行わなくても，by-wireのデモンストレーションは実行できます．しかし，タイム・トリガ型の利点である「バケツ・リレー的な通信」において，比較的容易に時間的制約を守れることを示すため，三つ目のノードにはFlexRay通信間のゲートウェイのような処理を担当させたいと思います．

以上のことを整理し，各ノードの役割をまとめると，表2のようになります．

● 処理内容に合わせてハードウェアを変更する

上記のように各ノードが処理を行うために，ハードウェアを少し改造する必要があります．

表2　各ノードの役割

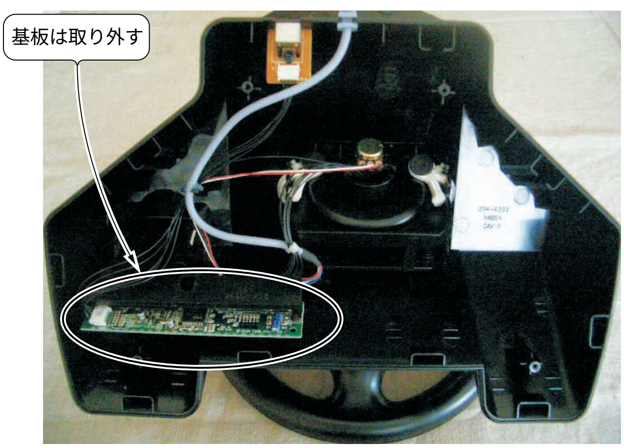
ノード	役　割	接続デバイス	処理内容
ノード1	各種情報入力	ゲーム・コントローラ	ゲーム・コントローラから操舵角やアクセル，ブレーキの各情報を読み出し，ディジタル信号に変換してFlexRayネットワーク上に情報を送信する
ノード2	機器（動作）制御	ステアリング用ラジコン・サーボ，モータ・コントローラ	FlexRayネットワークを介して操舵角信号，アクセル信号，ブレーキ信号を取得し，その信号に従ってステアリングやモータの制御をPWMで実現する
ノード3	FlexRay通信の時間同期およびゲートウェイ	とくになし	FlexRayネットワーク上のデータを受信し，データIDなどを変更して送信する



（a）正面



（b）上部から見たところ



（c）内部



（d）ステアリング部拡大

写真2　ゲーム・コントローラ

本ゲーム・コントローラのハンドル部分に付いているボタンは使用しない．操舵角情報はハンドルの回転ぐあいから，アクセルはハンドル右側の，またブレーキはハンドル左側のスライダ装置から取得することにした

1）ゲーム・コントローラの改造

ゲーム・コントローラからは，操舵角やアクセル，ブレーキなどの情報を入力します．これらの情報はON/OFF（あるいは‘1’，‘0’）といった単純な信号ではなく，アナログ入力（どれくらい回転したか，どれくらいアクセル/ブレーキが踏まれたかといった割合）が望ましいと考えます．そのため，ゲーム・コントローラのボタンによる指令ではなく，アナログによる比率の入力が可能なデバイスを利用したいと考えました．

今回は本ゲーム・コントローラのハンドル部分に付いているボタンは使用しません．操舵角情報はハンドルの回転ぐあいから取得し，アクセルはハンドル右側の，ブレーキはハンドル左側のスライダのような入力装置から取得することにしました（**写真2**）．

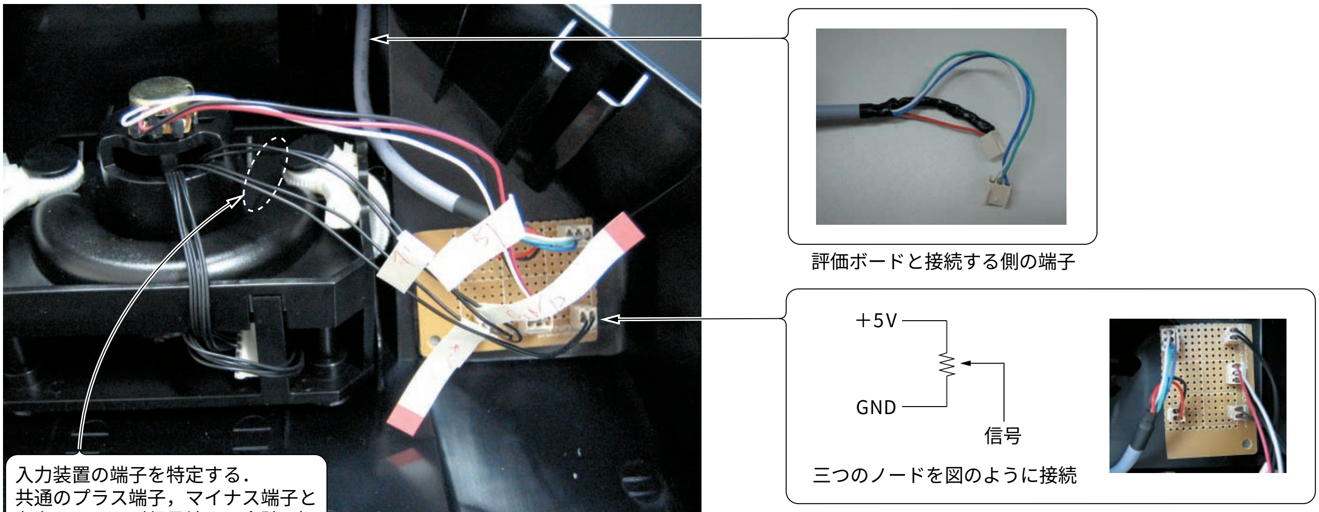


写真3　ゲーム・コントローラを改造
今回使用する信号に合わせて，ゲーム・コントローラを改造した

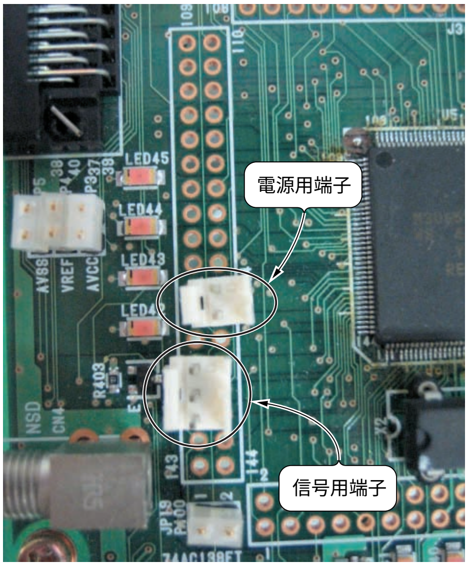


写真4　ノード1を改造
電源用と信号用の端子を取り付ける

ゲーム・コントローラの底をあけてみると，利用したい部分（ステアリング，アクセル，ブレーキ）にはいずれも可変抵抗器が付いていました．そこで，この可変抵抗器を利用してアナログ・データを入力することにしました．ゲーム機本体と本コントローラがどのような手順でデータをやりとりしていたかは不明ですが，今回はアナログ値を評価ボードで読み出したいだけなので，コントローラに付属していたボードは取り外します．

ステアリングの可変抵抗器は目視できますが，スライダ装置の可変抵抗器は目視できません．本コントローラに付属している制御基板には，ハンドル内部から信号ケーブルが接続されています．テスタなどを使ってこのケーブル端子からスライダ装置の端子を探し出します．端子には，スライダやボタンに電源を供給するプラス端子，マイナス端子のケーブルと，左右のスライダの入力端子があります．この合計4本を見つけ出せば，左右のスライダを利用できます．

次にゲーム・コントローラとゲーム機本体を接続するケーブルを調べてみると，赤色，黒色，白色，青色，緑色の合計5本が使われていることがわかりました．今回は三つの入力装置を扱いたいので，この5本のケーブルをすべて利用します．赤色ケーブルには＋5Vを，黒色ケーブルにはグラウンドを接続します．残りの3本は，ステアリング，アクセル，ブレーキの信号用ケーブルとします．最後に，ノード1（入力ノード）と接続できるようにケーブルを改造すればOKです（**写真3**）．

2）ノード1の改造

ノード1は，ゲーム・コントローラと接続できるように，電源端子および各信号用端子を取り付けます（**写真4**）．電源はマイコンのV_{cc1}端子（ピン番号132）とV_{ss}（グラウンド）端子（ピン番号130）につなぎます．また，各信号線はFlex Ray評価ボードのA-Dコンバータに接続します．今回，A-Dコンバータとしては評価ボードに搭載されているマイコン（M32C/85）に内蔵されたものを利用します（マイコンの端子名AN1，AN3，AN5）．このうち，AN1（ピン番号139）はステアリング情報の入力に，AN3（ピン番号137）はアク

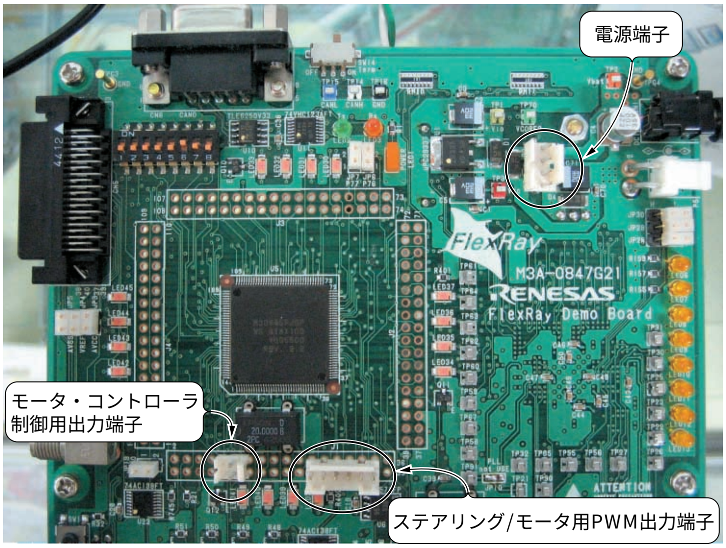


写真5　ノード2を改造
タイヤの回転に同期して処理を実行する機能を追加している．写真のボードにはホール・センサなどを接続するための端子が取り付けられている

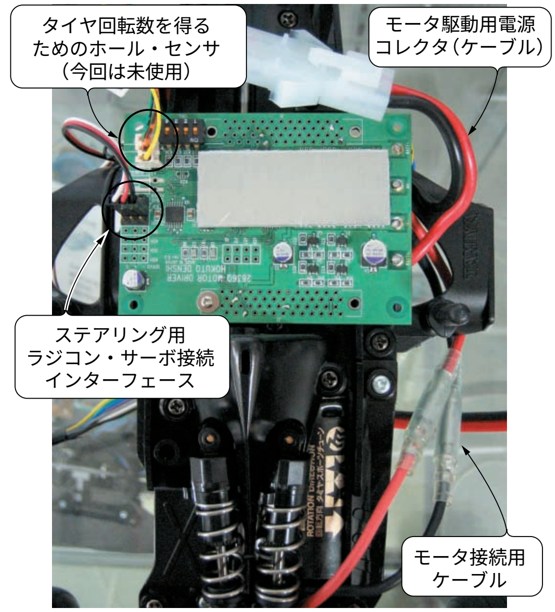


写真6　モータ・ドライバを改造

セル情報の入力に，AN5（ピン番号135）はブレーキ情報の入力に割り当てます．

3）ノード2の改造

ノード2は，FlexRay通信によって得られた情報をPWM信号に変換して機器を駆動します．ステアリング用のPWMは1本のI/Oを，モータ制御には正転/逆転用に2本のPWMと，さらにモータ・ドライバ制御用に2本のI/O出力を必要とします．評価ボード上のマイコンの端子$TA2_{OUT}$（ピン番号33）を前転モータ制御用に，$TA3_{OUT}$（ピン番号31）を後転モータ制御用に，$TA4_{OUT}$（ピン番号29）をステアリングのラジコン・サーボ（RCサーボ）用に割り当てます（**写真5**）．また，OUTC16（ピン番号12）とOUTC17（ピン番号11）はモータ出力制御用に割り当てます．

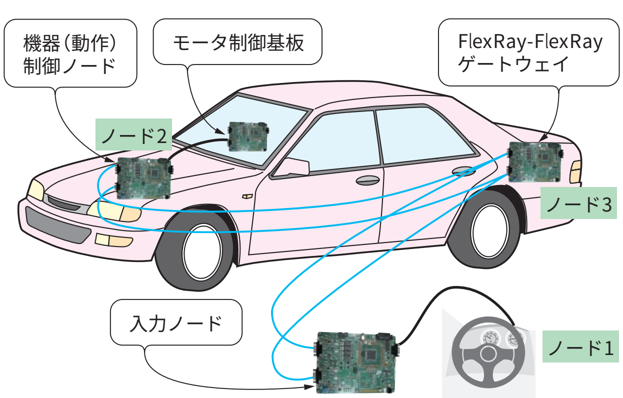


図4　ノード接続図
ノード1とゲーム・コントローラを，ノード2とラジコン（モータ・ドライバ）をそれぞれ接続する．これでハードウェアの構築は完了

4）モータ・ドライバの改造

北斗電子製のモータ・ドライバ基板には，通常，同社製のマイコン・ボードを接続するためのインターフェースが取り付けられています．今回はこのインターフェースを利用しないため，筆者らが利用したボードでは取り外しています．

本モータ・ドライバ基板上には，複数のラジコン・サーボ接続用端子が取り付けられています．なお，デモンストレーション・システムで用いるラジコン・サーボは3芯ケーブルで，信号，＋極，GND極の3極となっています．

また，本モータ・ドライバ基板では，基板に電源を供給するとラジコン・サーボにも電源が供給できるようになっているので，信号線に接続されている端子とマイコンのステアリング端子を接続するように配線します（**写真6**）．

次に，マイコンのモータ制御用信号線とモータ・ドライバのモータ制御用信号を接続します．モータ・ドライバ基板とラジコン用モータを接続する端子ケーブルを取り付けます．また，ラジコン用のバッテリと接続するためのコネクタ・ケーブルを取り付けます．

最後に，改造した各ボードを接続します（**図4**）．各ノードは，2チャネルで接続（二重化）します．

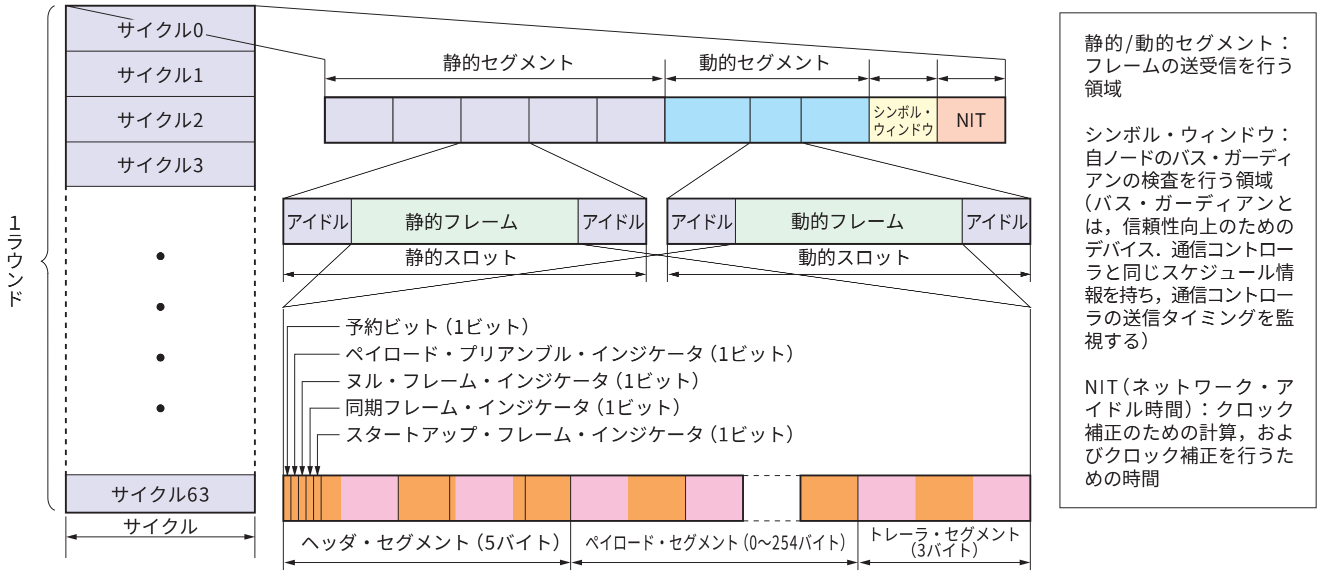
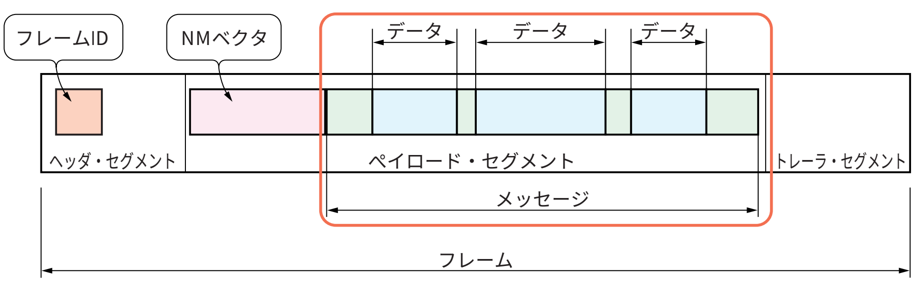


図5　ラウンド，サイクル，フレームの関係図

1ラウンドは最大64サイクルからなる．1サイクルは，静的セグメント，動的セグメント，シンボル・ウィンドウ，NIT（ネットワーク・アイドル時間）で構成される．静的セグメントはシステム設計時に決定した長さのデータ・フレームを扱うため，静的スロット長は固定．一方，動的セグメントは可変長のデータ・フレームを扱うため，動的スロット長も可変である．フレーム・フォーマットは静的フレーム，動的フレームともに，ヘッダ，ペイロード，トレーラという三つのセグメントで構成される．今回開発したFlexRay-NMは，ペイロード上にNM（ネットワーク管理）ベクタを搭載してネットワーク管理を行う．NMベクタは，ペイロード・セグメントの最初の0〜12バイトを割り当てられており，ユーザが定義するネットワーク管理を行う．これは静的セグメントで利用される

図6
データ，メッセージ，フレームの関係図

図の概念は，TT-COM，FlexRay-NMのミドルウェアで規定したもの．それぞれデータID，メッセージIDで管理されるが，これらのIDはTT-COMとFlexRay-NMの中でのみ利用され，送信データとしてやりとりされるわけではない．アプリケーションからはこの二つのIDを用いて操作するため，スロットやフレームのIDを意識する必要はない



3 ソフトウェアを実装する

FlexRay通信では，「フレーム」と呼ばれる単位で通信が行われます（**図5**）．フレームは静的セグメントと動的セグメントに含まれ，それが「サイクル」と呼ばれる単位で管理されます．さらに64個のサイクルが集まり，「ラウンド」という単位でFlexRay通信は管理されます．

● FlexRay通信の基礎知識

フレームは通信スロットごとに送受信され，それぞれフレームIDとスロットIDと呼ばれる識別子で管理されます．これらのIDはFlexRay通信の仕様で規定されています．

筆者らが開発したFlexRay通信ミドルウェアでは，さらにメッセージID（メッセージの種別を示す）とデータID（データの種別を示す）を規定しました（**図6**）．FlexRay通信ミドルウェアへは，これら論理的なID（メッセージID，データID）で指令を出します．以下に，データの関係を示します．

- **データ** ―― FlexRay通信でやりとりする最小単位．データの種別を示すID（データID）は，データの送信時および受信時に指定する．ttSendMessage，ttReceive Messageなどの API（application programming interface）で利用する．データIDは，どのメッセージ内のどの位置に格納されるかが関連づけられている
- **メッセージ** ―― 複数のデータから構成される．メッセージの種別を示すID（メッセージID）は，フレーム操作を行う場合に指定する．フレームとは1対1に関連づけら

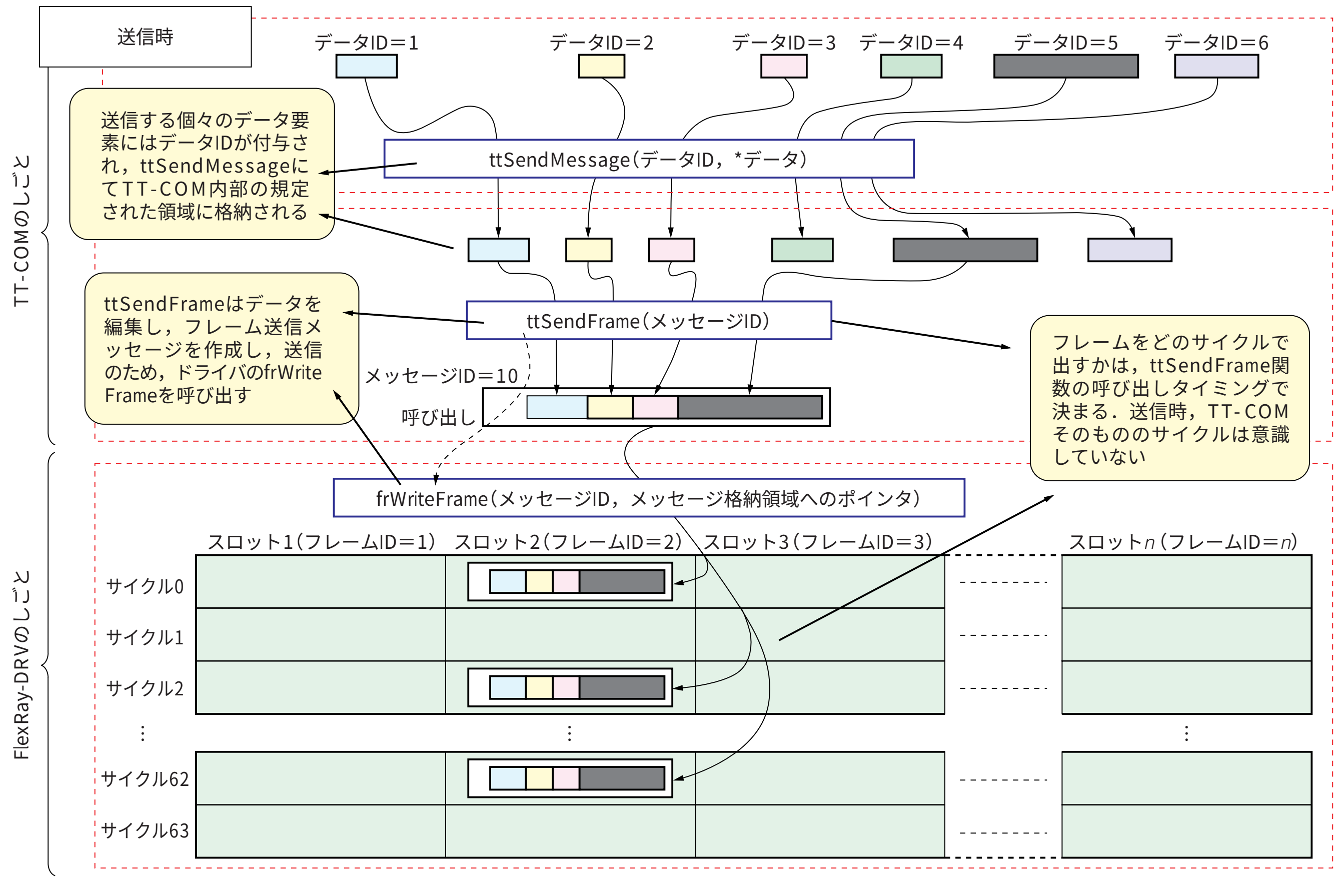


図7　送信時の各IDと利用するAPI

図は，送信処理要求（ttSendMessage）を発行した後のCOM内部での動作を示している．通常，アプリケーションはデータIDでデータを管理している．そのため，アプリケーションからはデータIDとデータの内容をttSendMessageに与えることにより，COM内部で送信単位であるフレームにパッキングされる（つまり，複数のメッセージが一つのフレームにまとめられる）．その後，フレーム単位で送信される．メッセージIDはCOM中で処理するためのIDであり，フレーム単位で管理する

れるが，フレームはメッセージ情報にフレーム・ヘッダなどの情報を保持する．メッセージIDは，どのフレームに含まれるかの関連情報を持っている

- **フレーム** ── ヘッダ・セグメント，ペイロード・セグメント，トレーラ・セグメントなどから構成される．フレームIDは，ヘッダ・セグメントの中に格納される
- **スロット** ── 通常，1スロットに1フレームが割り付けられ，スロットID＝フレームIDとなる（スロットはアイドル時間も含めた送受信データ）

上述したように，FlexRay通信ミドルウェアではデータIDとメッセージIDを利用して送信処理を指令します．図7と図8に，送信時と受信時の各IDと処理内容を示します．

● 通信条件を設定する

デモンストレーション・システムを実現するためのノード構成が決まると，おおむねソフトウェアで処理すべきことが定まってきます．今回のシステムでは，ノード1がコントローラからステアリングやアクセル，ブレーキの情報を読み出し，これらの情報を中継ノードであるノード3に伝送します．ノード3はノード1の情報を受信し，ノード2にその情報を転送します．最後にノード2は送られてきた情報をもとに，モータ制御やステアリング制御を実行します．

これらの通信処理を実現するため，通信タイミングやデータを決定します．今回は，通信条件を以下のように決めます．なお，今回のデモンストレーションでは，リアルタイム性や信頼性などについてとくに大きな制約を設けていません．以下に示す条件はあくまでも一例です．実際に車載システムを設計する場合，自動車メーカが指定する要件に合った通信条件を設定する必要があります．

今回の通信条件を決定するための基本的な制約としては，

1）1サイクル内でセンサ入力から制御までを完結させる

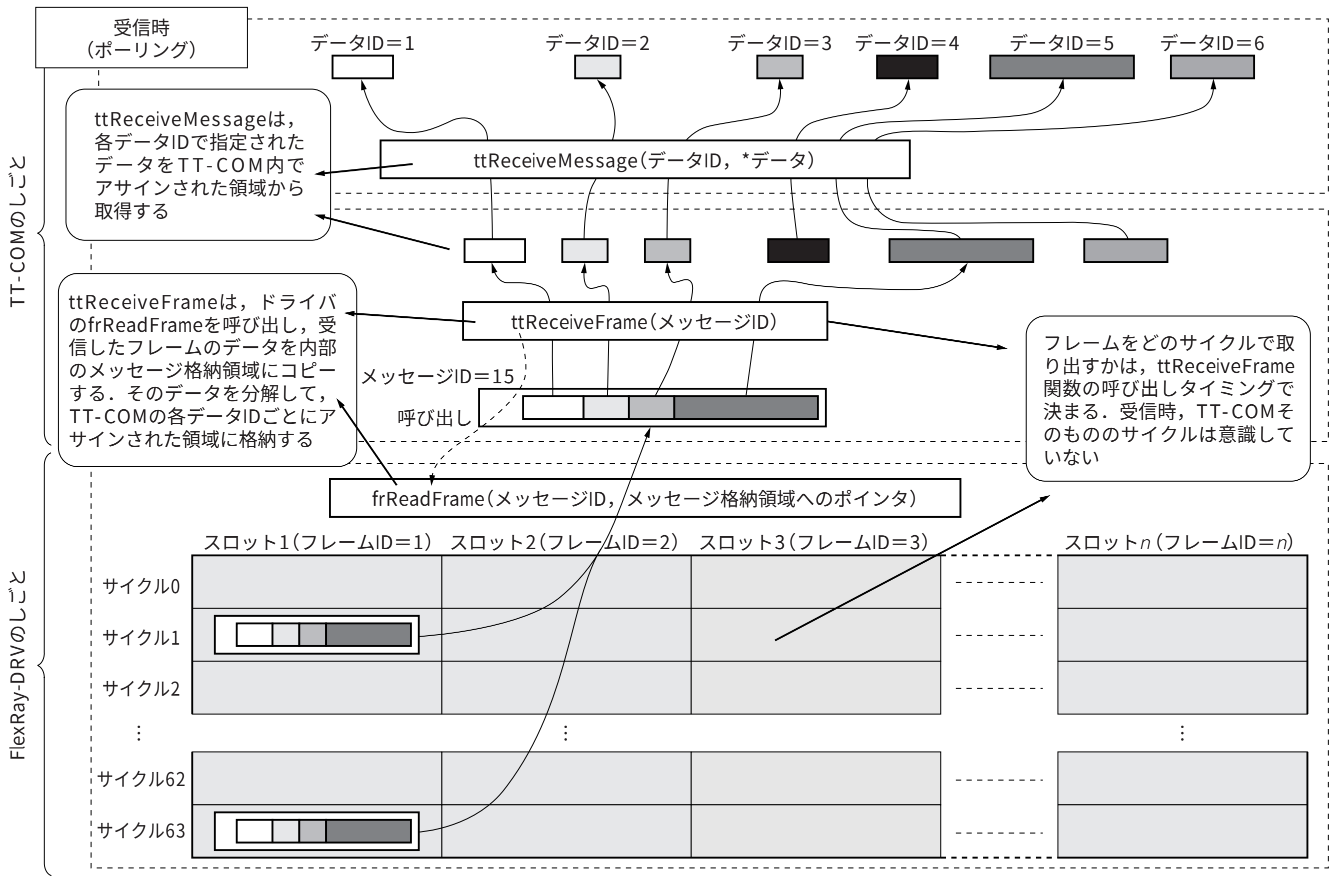


図8　受信時の各IDと利用するAPI

データを受信してからアプリケーションがデータ取得要求（ttReceiveMessage）を発行してデータを得るための動作を示している．ttReceiveFrameを呼び出すことにより，この関数からドライバのfrReadFrameが呼び出される．frReadFrameの実行タイミングを操作することにより，所望するタイミングでの受信が可能となる．COM内部ではメッセージ単位でデータを読み出し，ttReceiveMessageにてデータ単位でデータを取得する．

2）通信は全ノードを利用する

3）今後の拡張を考慮し，十分な送信データ長を確保する

　これらの制約を満たすため，以下の通信条件でシステムを実現することにします．なお，これらの指定はドライバ用OIL（OSEK Implementation Language）注6ファイル（frdrv.oil）で指定できます．下記の通信条件の中で［ ］で囲まれている文字はOILの指定項目名称を表します．

【通信条件】

- 1サイクル4000MT注7［MACROPERCYCLE］とする
- 1サイクルには20個の静的スロット［NUMBEROF STATICSLOTS］を割り当てる
- 1静的スロットは100MT［STATICSLOT］時間とする

　これらの条件を含め，1サイクル当たりの時間，静的スロットと動的スロットの数，1静的スロット当たりの時間（送信データ長），NIT（FlexRayデバイスの遅延補正時間）など，ほとんどすべての情報は，OILで静的に指定できます．詳しくは，OIL仕様を参照してください．

● 1スロットの送信データは通信コントローラに依存する

　今回は1サイクルを4000MT（＝160000μT；1MT＝40μT），静的スロット数を20，静的スロット長を100MTとしました．このような条件の場合，いったい1静的スロットで何バイトのデータを送信できるのでしょう？

　送信データ長は，スロット長とμTに大きく依存しています．μTは通信レートにより決定される時間単位です．

注6：OILは，OSEK/VDXが策定したシステム記述言語．OILに従ってオブジェクトの定義をファイル（OILファイル）に記述し，そのファイルをもとにシステム・ジェネレータと呼ばれるツールを使ってOSが必要とする情報（テキスト・ファイル）を生成する．なお，TT-OSでは，OSEK/VDXが策定したOIL仕様Version 2.5をベースに，筆者らが独自に拡張したフォーマットを規定した．

注7：MT（マクロティック）は，FlexRay通信の時間単位を示す．MTは複数のμT（マイクロティック）から構成される．μTは通信レートから算定され，1MTは1サイクル当たりのMT数と1サイクル当たりのμT数から算出される．今回のデモンストレーションでは，1サイクル＝4000MT＝160000μTで指定しているため，1MT＝40μTとなる．

通信レートはFlexRay通信コントローラによって異なるため，µTの値はすべてのFlexRayネットワークで同じ値であるとはいえません．ここでは，筆者らが仮定したFlex Ray通信コントローラを例に説明します．

図9のような構成をとる場合，1ビット送信するのに必要な時間（これを1ビット・タイムと呼ぶ，1ビット・タイム＝1÷伝送速度）は8µTになります．上記の通信条件では，1MTは40µTなので，1MT時間では5ビットの送信が可能となります．そして，1スロット時間である100MTでは500ビットを送信できることになります．通常，1バイトのデータを送信するためには，ストップ・ビットなどの情報を付加して合計10ビットが必要となります．すなわち，この仮想的な通信コントローラを用いた場合，静的スロット一つで50バイトのデータを送信できるというわけです．

なお，今回のデモンストレーションでは，理論上1スロット当たり100バイト送信できように設計しています．ペイロードのデータ長は64バイトのデータを扱うようにしました．

● 各ノードのタスクの実行

図10に，デモンストレーションで必要となる処理要求事項と通信条件から，本システムで利用するFlexRay通信のフレーム送受信とタスクの関係を示します．

今回は，ノード2（制御ノード）からほかのノードに伝送するデータはありません．かといって，ノード2はデータ送信を行わなくてもよいかといえば，そうではありません．ここで用いているFlexRay通信ソフトウェアは，ネットワーク管理をサポートしています．ネットワーク管理用の情報は各ノードの送信データ内に含まれているため（**図5**を参照），1サイクルに一度はネットワーク管理情報を送信しなくてはなりません．これらの条件から，次のようにスケジューリングします．

まず，サイクル開始（0MT）のタイミングでノード2のタスク1を起動します．このタスクはノード2の状態を判断し，適切な情報をネットワーク管理情報に反映させ，データを送信します．この情報はFlexRay通信を介して，各ノードに転送されます．

200MTになると，ノード1のタスク1が起動します．このタスクはノード2のネットワーク管理情報を参照し，ノード2の状態を把握します．ノード2は送信するべきほかの情報がないので，ノード2からの情報をもとに処理する必要はありません．このタスクは，ノード1に接続されているゲーム・コントローラから情報を取得し，アナログ・データをディジタル・データに変換し，保持します．

500MTになると，ノード1のタスク2が起動します．このタスクはタスク1が準備したコントローラの情報とノード1の状態から，コントローラ・データとノード1のネットワーク管理情報を送信データとして作成し，送信します．

800MTになると，ノード3のタスク1が起動します．このタスクは，ノード1のネットワーク管理情報を参照し，ノード1の状態を把握します．また，ノード1から送られてきたコントローラの情報を受信し，その情報を保持します．

1300MTになるとノード3のタスク2が起動します．このタスクは，ノード3のタスク1が保持しているノード1からのコントローラ情報を利用して，ノード2に送信するコントローラ情報を作成します．また，ノード3の状態からノード3のネットワーク管理情報を送信データとして作成し，送信します．

1800MTになると，ノード2のタスク2が起動します．このタスクはノード3から送信されたデータをもとに，モータ制御およびステアリング制御を行います（ノード2もノード1が送信するデータを受信できるが，ノード3を中継ノードとしているため，あえてノード1が送信するデータを利用していない）．

● 送受信タイミングに注意が必要

本システムのソフトウェア構成は，各ノードとも非常にシンプルです．すべてのノードが送信用（データ取得用）と

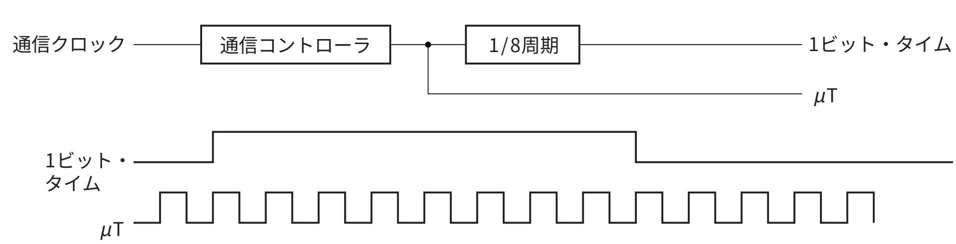


図9 仮想FlexRay通信コントローラ（クロック部）

図のような構成をとる場合，1ビット送信するのに必要な時間（これを1ビット・タイムと呼ぶ）は8µTとなる

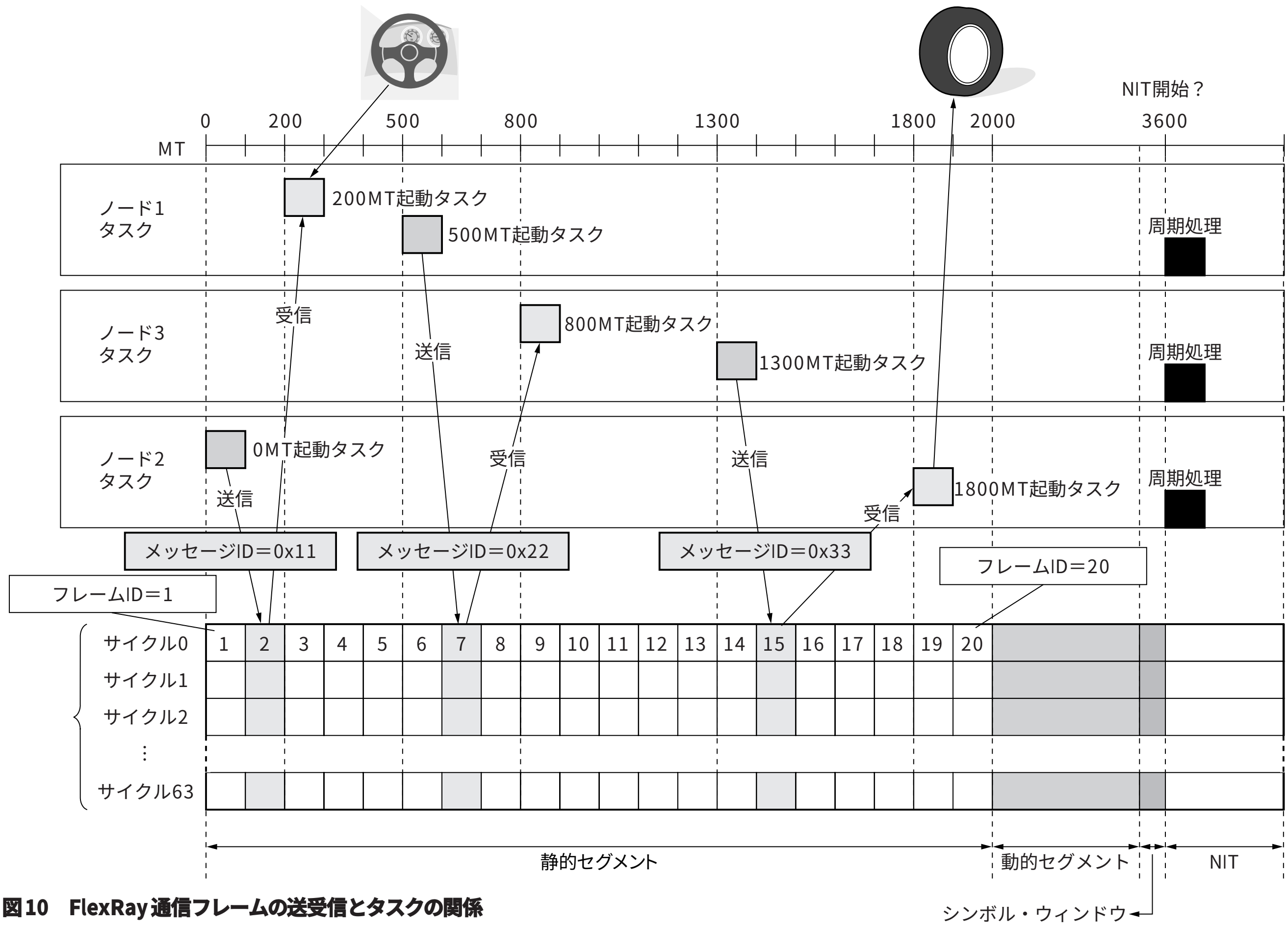


図10　FlexRay通信フレームの送受信とタスクの関係
サイクル開始（0MT）のタイミングでノード2のタスク1を起動する

受信用（モータ制御およびステアリング制御）の二つのタスク注8で構成されています．この二つのタスクは，FlexRayネットワークの時間（グローバル時間）を基準に起動します．これは，これらのタスクがネットワークにおける送受信に関係するためであり，送受信に関係ないタスクであればグローバル時間を基準にして起動する必要はありません（なんらかのイベントをトリガ条件にしてタスクが起動されればよい）．

FlexRay通信はスロット単位でデータを送受信します．静的フレームを利用した場合，通常，特定のデータは特定のスロットを利用して送受信します．このスロットへのデータ書き込みは，当該スロットが送信される少し前（この時間パラメータは利用するマイコンに依存する）までにデータを設定すれば，そのデータが送信されます．データを設定しないと，以前のデータがそのまま送信されます．したがって，新しいデータを送信したい場合，送信される直前に送信データを送信スロットに設定すればよいわけです．つまり，スロットにデータ設定が完了しなくてはならない時間から逆算してタスクを起動させれば，問題ありません．

受信についても同じです．受信が完了する前にタスクが起動し，データを読み出しても，以前のデータしか取得できません．新しい情報を受け取りたい場合は，受信が完了した直後にタスクを起動させてデータを取得するのがよいと思います．FlexRay通信では，必要とするデータがいつ送信されるかが静的に決められているため，受信完了直後にタスクを起動させることができます（**図11**）．

注8：正確には，すべての通信サイクルの最後にTT-COMやFlexRay-NMに対して必要な処理を行う．TT-COMが提供する関数を呼び出してミドルウェアに必要な処理を実行する．具体的には，FlexRay-NMの周期処理関数frTickNM( )やTT-COMの周期処理関数ttTickCOM( )を実行する．また，同期情報を取得する必要がある．今回のデモンストレーションでは，8msごとに起動するアラームを用いて周期タスクを起動している．その周期タスク内で外部時間との同期状態を得るためのTT-OSのAPIであるttGetSyncStatus( )をコールする．

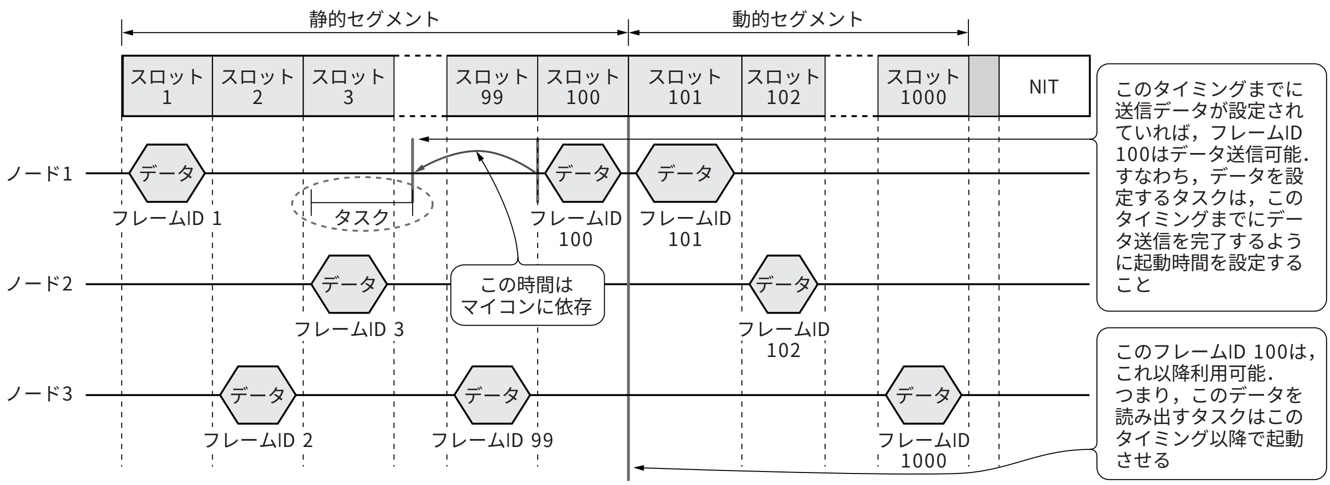


図11　送信タスクと受信タスクの起動タイミング

図の送信例では，ノード1にフレームID 1，100，101の送信権がある．図の灰色の点線で囲まれた部分のタスクは，指定時間までに送信データを作成して送信すればよい．しかし，時間までにデータを設定できたかどうかを確認するには，タスクに細工を施す必要がある．TT-OSでは，タスクの完了を確認するデッドライン監視機能を提供している．図のタスクは，データ設定後にほかの処理を行うのではなく終了するように設計しておけば，TT-OSのデッドライン監視機能で正しくデータ送信処理が完了しているか否かを確認することができる

● ミドルウェアを利用してFlexRay通信を実行する

FlexRay通信ミドルウェアの利用法を簡単に説明します．まず，FlexRay通信の初期化シーケンスについて述べます．電源投入後，TT-OSがスタートアップします．OSをフックした後，TT-OS，FlexRay-NM，TT-COMのAPIを呼び出します．

- frInitNM()：FlexRay-NMモジュールの初期化
- ttInitCOM()：TT-COMモジュールの初期化
- ttmInitTTM()：TTMモジュールの初期化
- frSetSleepRequest：スリープ要求の通知

すると，OSが稼働して，FlexRay調整モードに入ります．調整モードにおいてネットワークと同期が完了するとFlexRay通信を行える状態になります．これで初期化は終わりです．

次にデータ受信処理ですが，これはTT-COMのAPIであるttReceiveFrame(メッセージID)を利用してフレームを受信し，メッセージを取得します．このとき，アプリケーションはまだデータを取得していません．FlexRay-NMはNMメッセージ(ネットワーク管理情報)を取得して，TT-COMはアプリケーションにデータを渡す準備が完了します．次に，ttReceiveMessage(データID，格納バッファ・ポインタ)を利用して，メッセージからアプリケーションが指定したデータを抜き出し，指定されたバッファに格納します．

データ送信処理の場合，ttSendMessage(データID，格納バッファ・ポインタ)を用います．ttSendMessageに含まれるデータIDの情報に従って指定データをメッセージに格納します．その後，ttSendFrame(メッセージID)を用いてメッセージからフレームを作成します．フレームに格納されたデータをスロットに格納した後，TT-COMはヘッダ・セグメントとトレーラ・セグメントを作成し，FlexRay-NMはNMベクタに自タスクの情報を設定します．そして，フレームをスロットに格納して，送信処理を実行します．

● スケジュール・テーブルを記述するOILファイル

今回用いたFlexRay通信用ソフトウェアは，OILを利用してタイム・トリガ型のスケジュールを記述します．今回のデモンストレーション・システムでは，FlexRay通信を利用するにあたり，各ノード用に三つのOILファイル(本誌付属のCD-ROMに収録)が必要となります(既存OS用OILファイルを除く)．

- ttm.oil：TTM用OILファイル．スケジュール・テーブルを含む(**リスト1**)
- ttcomfrnm.oil：TT-COM用とFlexRay-NM用のOILファイル
- frdrv.oil：FlexRay-DRV用OILファイル

将来的には，FlexRayネットワークに接続されるすべて

のノードを対象とした，スケジュールを規定するためのツールを用意する予定です(p.50のコラム「ツール・チェーンとOIL」を参照)．FlexRay通信用ソフトウェアでは，そのツールによってノードごとに出力された情報からノード専用のスケジュール・テーブルを作成します．筆者らは，ノード情報はOILを用いて記述するべきであるとの考えから，スケジュール・テーブルもOILにて記述する方法を採用しました．つまり，ノード(ECU)単位より粒度が細かいものは，OILで記述するべきであるという判断です．

● ハード，ソフトが決まったら通信の正しさを確認する

さて，ハードウェアも決定し，ソフトウェアの構成や動作内容が規定されると，それに合ったソフトウェアを作成することになります．今回のデモンストレーションでもっとも動作が見えにくいのはFlexRay通信だと思います．では，FlexRayによる通信が正しく行われているか否かをどのように確認すればよいのでしょう？

今回筆者らは，FlexRay通信の確認にサニー技研製のFlexRayバス・モニタ(S810-F1 FlexRay Station)を利用しました．本バス・モニタを用いると，通信を行っているかどうかが不明な状況でも，FlexRay通信コントローラがなんらかの動作を行っているかを判断できます．これは，このバス・モニタにバス上になんらかの情報が出ているかどうかを簡易的に調査する機能が備わっているからです．通信が行われているようであれば，正しい通信パラメータを設定し，通信レートやフレーム長などをチェックします．

リスト1　ノード1のスケジュール・テーブル記述方法(ttm.oil)

```
#include "implementation.oil"

CPU current {
    FLEXRAYNETCONFIG flexraynet{
        ROUND = 64;
        CYCLETIME = 4000;
        MAXCORRECTIONTIME = 30;
    };

    TTMCONFIG  ttmconfig {
        NONSYNCCYCLETIME = 4400;
        CLOCKSYNC = SMOOTH;
        CYCLESYNC = SMOOTH;
        TTMSTARTUPHOOK = TRUE;
        TTMERRORHOOK = TRUE;
        NETWORKNOSYNCHOOK = TRUE;
        NETWORKSYNCHOOK = TRUE;
    };

    TTMACTION ttmact_recvid11 {
        CYCLE = 1;
        FIRSTCYCLE = 0;
        ACTIONTIME = 200;
        ACTION = ACTIVATETASK {
            TASK = task_a1;
        };
    };

    TTMACTION ttmact_sendid22 {
        CYCLE = 1;
        FIRSTCYCLE = 0;
        ACTIONTIME = 500;
        ACTION = ACTIVATETASK {
            TASK = task_a2;
        };
    };

    TTMACTION ttmact_tick {
        CYCLE = 1;
        FIRSTCYCLE = 0;
        ACTIONTIME = 3600;
        ACTION = ACTIVATETASK {
            TASK = task_a5;
        };
    };
};
```

(FLEXRAYNETCONFIGへの注記)
● FlexRayの通信1ラウンド・サイクル数＝64
● 1サイクルの時間＝4000MT
● 最大補正時間＝30μs

(TTMCONFIGへの注記)
● ローカル・タイマのベース時間＝4400MT
● 時間同期，サイクル同期はスムース同期を選択
● 各種フックを有効にする

(ttmact_recvid11への注記)
ノード1のタスク1の記述方法(スケジュールを含む)
● 最初の起動はサイクル0(最初のサイクル)で周期が1なので，タスク1は毎サイクル実行
● サイクル内の200MTで処理される．処理の内容はtask_a1の起動要求

(ttmact_sendid22への注記)
ノード1のタスク2の記述方法(スケジュールを含む)
● 最初の起動はサイクル0(最初のサイクル)で周期が1なので，タスク2は毎サイクル実行
● サイクル内の500MTで処理され，処理の内容はtask_a2の起動要求

(ttmact_tickへの注記)
ノード1のタスク3の記述方法(スケジュールを含む)
● 最初の起動はサイクル0(最初のサイクル)で周期が1なので，タスク3は毎サイクル実行
● サイクル内の3600MTで処理され，処理の内容はtask_a5の起動要求

注：TT-COM(FlexRay-NM)を利用する場合，サイクルの終盤(3600MT付近)でサイクルごとに必要な処理をコールする必要がある

参考・引用*文献

(1) OSEK/VDX；OSEK/VDX Time-Triggered Operating System Version 1.0, July 2001.
(2) OSEK/VDX；OSEK/VDX Operating System Version 2.2.1, Jan. 2003.
(3) 大熊繁監修；車載LANとその応用，トリケップス，2005年6月.
(4) カーエレクトロニクス テクニカルターム 特別編集版 AD Special, 日経BP社，2005年3月.
(5) 服部博行；FlexRay実現に向けた時間駆動型通信リアルタイムOSを開発(前編)，Design Wave Magazine，2006年4月号，pp.115-121.
(6) ルネサス テクノロジ；M32C/85グループ ハードウェアマニュアル Rev.1.03，2005年7月.
(7) ルネサス テクノロジのWebサイト，http：//japan.renesas.com/
(8) 北斗電子；2636Q MOTOR DRIVER取扱説明書.
(9) 原田了；自動車のメカニズム，日本実業出版社，2000年2月.

はっとり・ひろゆき
(株)ヴィッツ

<筆者プロフィール>

服部博行．自分の興味のあるものは自分で作りたがるやっかいな性格．日曜大工，ビール作りなどで自分の時間を楽しんでいる．最近は，2人乗りの車でドライブを楽しみ，家族からは白い目で見られているが，本人はまったく気にしていない．